Desenvolvimento de um circuito aritmético a partir da sua especificação em Handel-C

Pedro Almeida, Manuel Almeida

Resumo **- Este artigo descreve um circuito baseado numa FPGA (Field Programmable Gate Array) que implementa quatro operações aritméticas (+, -, *, /) e que interage com um monitor e com um rato ligados à FPGA. A especificação do circuito foi feita em Handel-C, que é uma linguagem de alto nível desenvolvida pela Celoxica. Esta especificação em Handel-C foi verificada no ambiente *Celoxica DK1 Design Suite* e compilada para um ficheiro EDIF que por sua vez foi convertido num "*bitstream*" para FPGA no ambiente *Xilinx ISE 5.1*. O circuito especificado foi testado na FPGA da família Spartan-II XC2S200 que é o componente reconfigurável principal da placa RC100 fornecida pela Celoxica. Este artigo mostra algumas das potencialidades do Handel-C e DK1, tal como a possibilidade de compilação para o código VHDL.**

Abstract **- This paper describes an FPGA-based circuit that implements four arithmetical operations (+, -, *, /) and interacts with a monitor and a mouse attached to the FPGA. Functionality of the circuit has been described in Handel-C, which is a system-level specification language, developed by Celoxica. The Handel-C specification was verified in Celoxica DK1 design suite and translated to EDIF file that was converted in Xilinx ISE 5.1 environment to a bitstream for FPGA. The designed circuit was tested in FPGA Spartan-II XC2S200, which is a primary reconfigurable component of RC100 board supplied by Celoxica. The paper demonstrates some additional opportunities of Handel-C and DK1, such as generation of a synthesizable VHDL code.**

I . INTRODUÇÃO

O trabalho aqui descrito vem no seguimento da disciplina SDA (Sistemas Digitais Avançados) que vai ser utilizado como um componente do projecto final. O leccionado em SDA forneceu as bases necessárias para a linguagem de descrição de hardware (VHDL) assim como a utilização do ambiente da Xilinx.

A. Características principais da placa RC100

A placa RC100 tem como componente reconfigurável principal a FPGA da família Spartan-II da Xilinx com 200.000 portas lógicas.

Esta FPGA tem ligação directa com :

- Oscilador de cristal de 80 MHz [1];
- Dois blocos de memória RAM síncrona estática (SSRAM) com 256K palavras de 36bits cada;
- Flash RAM de tamanho 64 Mbits;
- Vídeo DAC (VGA 24-Bits);
- Descodificador da entrada de vídeo;
- Porta PS/2;
- LED's;
- 2 *displays* de 8 segmentos;
- Um barramento de expansão.

É ainda possível aceder à porta paralela através do CPLD.

B. Especificação do projecto

O projecto consiste no desenvolvimento de um programa na linguagem Handel-C que permite fazer os cálculos básicos apresentando-os num ecrã de um monitor VGA como pode ser visto na fig. 1.

Nas duas primeiras janelas são apresentados os operandos sendo o resultado da operação escrito na terceira. O controlo do programa é efectuado pelo rato que pode ser usado da seguinte forma:

- Quando posicionado o cursor do rato no interior de uma das janelas dos operandos, é indicado que a janela está seleccionada através da alteração da cor do seu limite para vermelho, podendo agora ser alterado o seu valor numérico. Para tal, o valor do operando é incrementado ou decrementado de uma unidade, consoante se clique com o botão esquerdo ou direito do rato respectivamente.
- A apresentação do resultado é obtida com um clique em qualquer um dos botões do rato quando o cursor deste se encontrar no interior da janela correspondente ao resultado.
- A escolha da operação sobre os operandos é feita através do clique de um dos botões do rato, quando o cursor deste se encontra sobre o sinal da operação podendo ser escolhidas as operações soma (+), subtracção (-), multiplicação (*) e divisão (/).
- É ainda possível mover o conjunto das janelas pela área visível do ecrã. Para tal, posiciona-se o cursor no limite de uma das janelas e mantendo pressionado o botão esquerdo do rato desloca-se as janelas para a posição pretendida.
- Estes dois periféricos (rato e monitor) estão ligados às respectivas portas existentes na placa RC100. Nesta placa existem ainda dois *displays* de 8 segmentos, os

quais estão, neste caso, a mostrar em tempo real as coordenadas da posição actual do cursor do rato no monitor (ver fig. 1).

Fig. 1 – Ligação dos periféricos

II . ESTRUTURA DO PROGRAMA EM HANDEL-C

A. Breve alusão à linguagem Handel-C

Handel-C é uma linguagem para especificar circuitos electrónicos ao nível de sistemas [2]. Esta linguagem utiliza uma estrutura semelhante à linguagem ANSI-C. Existem no entanto novas expressões que facilitam a escrita de programas a ser implementados em lógica reprogramável, como por exemplo:

- width( ) - devolve o número de bits de uma variável;
- @ - concatenação;
- \\ - elimina bits menos significativos (*Less Significant Bits* - LSB);
- <- - extrai LSB's;
- par {...} - execução em paralelo;
- seq {...} - execução sequencial;
- delay - atraso de 1 ciclo de relógio.

A definição de variáveis, tem agora a possibilidade de serem declaradas com o tamanho pretendido (em numero de bits).

O ambiente utilizado para a programação em Handel-C (*DK1 Design Suite* da Celoxica) contem uma livraria (RC100.lib) com macros (*drivers*) para utilização dos vários dispositivos e interfaces da placa RC100 [3].

B. Estrutura básica do programa

O programa é constituído por um grupo de blocos principais executados em paralelo (ver fig. 2) para que se tenha disponível o mais rápido possível os resultados dos vários blocos.

Na fase de "Inicialização", são inicializadas as variáveis que o necessitem. De seguida são executados em paralelo os vários blocos principais.

No "Processamento de dados", é calculado o resultado da operação, assim como o valor dos vários dígitos de todos os números.

Na "Validação para escrita dos dígitos no ecrã" são testadas as condições para saber se o *pixel* que está a ser iluminado pertence a algum dígito ou sinais.

O bloco denominado por "Movimento das janelas" contém o código necessário para se poder mover as janelas dentro dos limites da zona visível do ecrã.

O bloco "Desenho das janelas" é responsável pelo desenho das janelas numa dada posição da zona visível do ecrã.

Na "Alteração da operação" é verificado se o utilizador pretende alterar a operação entre os operandos e actualizar o respectivo sinal.

O Bloco "Definição da cor de cada *pixel*" é o que envia o sinal RGB (*Red*, *Green*, *Blue*) de cada *pixel* para o monitor dependendo das condições actualizadas pelos blocos anteriormente descritos.

Por último, o bloco "Escrita das coordenadas do rato nos *displays*" permite ajustar o valor actual da posição do cursor do rato para uma gama de 16 posições e envia os respectivos valores para os *displays*.

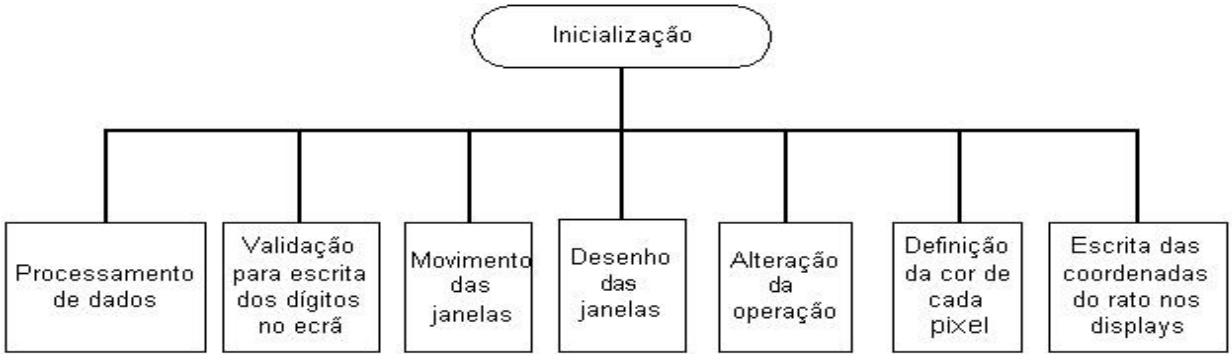

Fig. 2 – Blocos principais do programa

III . CONTROLO DOS DISPOSITIVOS

A. Controlo dos displays da placa

Para o controlo dos *displays*, foram criadas 2 ROM's para conversão de código hexadecimal para código BCD da seguinte forma:

```
rom unsigned HexDisplayEncode1[16] =
{0x3f,0x6,0x5B,0x4F,0x66,0x6D,0x7D,0x7,0x7F,0x6F
                ,0x77,0x7C,0x39,0x5E,0x79,0x71};
rom unsigned HexDisplayEncode2[16] =
{0x3f,0x6,0x5B,0x4F,0x66,0x6D,0x7D,0x7,0x7F,0x6F
                ,0x77,0x7C,0x39,0x5E,0x79,0x71};
```

Para mostrar as coordenadas (x,y) do rato nos *displays* foram invocadas as funções *RC100Set7Seg1* e *RC100Set7Seg2* da livraria RC100.lib que permitem enviar para os *displays* os sinais de cada segmento como se pode ver no código a seguir [4]. Como cada *display* mostra no máximo até "F" ($F_{hex}=15_{dec}$), o ecrã que tem aproximadamente 512x512 *pixels*, este foi dividido em 16 partes iguais com 32x32 *pixels* cada. Para tal foi dividido o actual valor da posição do cursor (x e y com 10 bits cada) por 32, eliminaram-se os 5 bits menos significativos (\\) e utilizaram-se os 4 bits menos significativos dos 5 resultantes da operação anterior (<-).

```
Par
{
 RC100Set7Seg1(0@DisplayOutput1);
 RC100Set7Seg2(0@DisplayOutput2);
 DisplayOutput1 =
    HexDisplayEncode1[(px\\5) <- 4];
 DisplayOutput2 =
    HexDisplayEncode2[(py\\5) <- 4];
}
```

B. Envio de sinais para o ecrã do monitor

Basicamente, o cinescópio do monitor envia raios catódicos para o ecrã fazendo um varrimento horizontal e vertical. É portanto necessário saber qual a posição exacta (x,y) do *pixel* que está a ser "iluminado" para assim definir qual a cor que se pretende que ele tome.

Para tal, existe na livraria RC100.lib, a estrutura *RC100_VGA_DRIVER* que contém as variáveis *ScanX* e *ScanY* que indicam qual o *pixel* que está a ser "iluminado". Existe ainda a macro *RC100RGB(red, green, blue)* que envia a informação da cor desse *pixel* em causa para o monitor no formato RGB - 24Bits.

É de salientar que a posição (x,y) devolvida pelo *driver* é actualizada a cada ciclo de relógio, sendo portanto importante garantir que o bloco responsável pela actualização do sinal RGB tenha a cada novo ciclo de relógio a informação da cor do *pixel* actual disponibilizada.

C. Desenho e movimento do cursor no ecrã

A posição do cursor no ecrã é controlada segundo a posição do rato. Essa posição é dada pelo sinal *PointerX* e *PointerY*, pertencentes à estrutura *RC100_PS2_MOUSE* presente na livraria RC100.lib

O desenho do cursor, neste caso, é definido por uma matriz 1x256 mas, para melhor compreensão, é vista como uma matriz de 16x16 contendo cada elemento a indicação da cor do *pixel* para o desenho do cursor como se pode ver no código a seguir. Cada posição pode conter:

- 0 – Transparente
- 1 – Branco (Interior do cursor)
- 2 – Preto (Limite do cursor)

```
rom unsigned 2 Pointer[256] =
{
   2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
   2,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
   2,1,2,0,0,0,0,0,0,0,0,0,0,0,0,0,
   2,1,1,2,0,0,0,0,0,0,0,0,0,0,0,0,
   2,1,1,1,2,0,0,0,0,0,0,0,0,0,0,0,
   2,1,1,1,1,2,0,0,0,0,0,0,0,0,0,0,
   2,1,1,1,1,1,2,0,0,0,0,0,0,0,0,0,
   2,1,1,1,1,1,1,2,0,0,0,0,0,0,0,0,
   2,1,1,1,1,1,1,1,2,0,0,0,0,0,0,0,
   2,1,1,1,1,1,1,1,1,2,0,0,0,0,0,0,
   2,1,1,1,1,1,2,2,2,2,2,0,0,0,0,0,
   2,1,1,2,1,1,2,0,0,0,0,0,0,0,0,0,
   2,1,2,2,1,1,2,0,0,0,0,0,0,0,0,0,
   2,2,0,0,2,1,1,2,0,0,0,0,0,0,0,0,
   2,0,0,0,2,1,1,2,0,0,0,0,0,0,0,0,
   0,0,0,0,0,2,2,0,0,0,0,0,0,0,0,0
}with { block = 1 };
```

É importante notar que as ROM's só podem ser acedidas por uma única parte do programa no mesmo ciclo de relógio.

Para se desenhar o cursor no ecrã, é necessário saber qual o *pixel* que o monitor está presentemente a "iluminar" e através de condições, determinar se o *pixel* em causa pertence a um *pixel* do cursor e, nesse caso, indicar qual a respectiva cor. Para determinar essas condições, utilizaram-se *shared expressions* que permitem devolver a várias partes do programa o resultado de uma dada condição como consta no código seguinte:

```
shared expr PointerValue =
Pointer[((sy-py)<-4)@((sx-px)<-4)];
shared expr PointerTransparent=
                                  PointerValue!=0;
shared expr PointerBlack = PointerValue==2;
shared expr PointerColour =
                  PointerBlack ? Black : White ;
macro  expr InPointer =
(sx>px && sy>=py && sx<=(px+16) &&
              sy<(py+16) && PointerTransparent);
```

IV. INTERFACE AO NÍVEL DO SOFTWARE

A. Escrita de algarismos no ecrã

Para escrever algarismos no ecrã é necessário, tal como no caso do cursor, construir uma matriz para determinar a forma de cada algarismo. Neste caso, foram construídas matrizes de 16x10 para cada algarismo e sinais como se

pode ver no exemplo, do numero "8", no seguinte código:

```
rom unsigned 1 oito[1][160]=
       {
               0,0,1,1,1,1,0,0,0,0,
               0,1,1,1,1,1,1,0,0,0,
               1,1,1,1,1,1,1,1,0,0,
               1,1,1,0,0,1,1,1,0,0,
               1,1,1,0,0,1,1,1,0,0,
               0,1,1,1,1,1,1,0,0,0,
               0,0,1,1,1,1,1,0,0,0,
               0,1,1,1,1,1,1,1,0,0,
               1,1,1,0,0,1,1,1,1,0,
               1,1,0,0,0,0,1,1,1,0,
               1,1,0,0,0,0,1,1,1,0,
               1,1,0,0,0,0,1,1,1,0,
               1,1,1,1,1,1,1,1,1,0,
               0,1,1,1,1,1,1,1,0,0,
               0,0,1,1,1,1,1,0,0,0,
               0,0,0,0,0,0,0,0,0,0,
       };
```

A escrita dos dígitos é feita baseada em condições. Por exemplo, para se escrever o referido dígito "8" a preto, colocando o primeiro *pixel* na posição (x,y) = (30,20) procede-se da seguinte forma:

- Verificar se $20 \leq ScanY < (20+16)$
- Quando ScanX = 30 , durante 10 ciclos de relógio consecutivos, verificar cada bit da matriz correspondente à posição (ScanX-30 , ScanY-(20+16)) e consoante o respectivo bit for :
  - '1' - enviar o sinal RGB = 0x000000 (Preto)
  - '0' - enviar o sinal RGB por defeito

Para a escrita de um número com vários dígitos, é conveniente que em cada varrimento horizontal Y sejam escritas todas as respectivas linhas (*Y-Yoffset*) de todos os números, sendo para o exemplo anterior Y = ScanY e *Yoffset* = 20.

Encontra-se a seguir um código possível em Handel-C para o exemplo anterior mas escrevendo 3 oitos seguidos a cor preta num fundo branco, sendo "oito" a variável definida no código anterior.

```
#define Preto      0x000000
#define Branco     0xffffff
unsigned 24 cor;
unsigned 2 i;
unsigned 4 j;
RC100_VGA_DRIVER Video;
…
do
{
 if((Video.ScanY>=20)&&(Video.ScanY<(20+16)))
   {               //16-> altura do dígito
    if (Video.ScanX==30)
    {
     i=3; //nº de dígitos seguidos
     while (i!=0)
     {
      j=0;
      while (j<10) //10->largura do digito
      {
       par
       {
        cor=oito[1][k+j] ? Preto : Branco;
        j++;
       }
      }
      i--;
     }
      k+=10;
    }
    else
     cor=Branco;
 }
 else
   par
   {
    k=0;
    cor=Branco;
   }
}while(1);
```

Como se pode ver no exemplo acima descrito, a posição dos algarismos no ecrã depende apenas das coordenadas (x,y)=(30,20). Substituindo os valores 30 e 20 destas coordenadas por valores genéricos x e y respectivamente, é possível escolher qualquer posição do ecrã para a escrita dos dígitos.

B. Processamento dos dados

Para o cálculo dos resultados, existe uma variável (*op* com o tamanho de 2 bits) que é actualizada quando o utilizador clica com o rato no sinal da operação. Dependendo da operação pretendida, a variável *op* toma os seguintes valores:

- 0 – Subtracção
- 1 – Adição
- 2 – Divisão
- 3 – Multiplicação

Fazendo um teste à variável *op*, é escolhida a operação a efectuar com os operandos.

No caso da subtracção, o programa foi implementado de forma o resultado ser sempre positivo. Quanto à divisão, uma vez que o compilador cria um código demasiado "pesado", foi criada uma função com um simples algoritmo de divisão como descrito no código seguinte.

Para que uma função possa ser invocada por mais que uma parte do programa no mesmo ciclo de relógio, esta deve ser definida com o parâmetro *inline* :

```
inline unsigned 8 div_8(unsigned 8,op_1,
                                  unsigned 8 op_2)
{
   unsigned 8 quo,resto;
   quo=0;
   resto=op_1;
   while(resto>=op_2)
   {
      resto=resto-op_2;
      quo++;
   }
return(quo);
}
```

As operações de adição e multiplicação, foram efectuadas da forma usual como mostra o seguinte código:

```
if (op==0) // Cálculo do resultado
 {
  if (op_1>=op_2)
    res=adju(op_1,width(res))-
                          adju(op_2,width(res));
  else
    update_res=0;
 }
 else
   if (op==1)
     res=adju(op_1,width(res))+
                          adju(op_2,width(res));
```

```
else
  if (op==2)
    if (op_2!=0)
      res=0x00 @ div_8(op_1,op_2);
    else
      update_res=0;
  else
    if (op==3)
      res=adju(op_1,width(res))*
                          adju(op_2,width(res));
    else
      delay;
```

V. COMPILAÇÃO DE HANDEL-C PARA VHDL

Usualmente, os programas em Handel-C são compilados para EDIF, sendo apenas necessário utilizar o ficheiro de saída EDIF no Xilinx ISE para a criação do *bitstream*, o qual é enviado directamente para a FPGA.

Para que o programa escrito em Handel-C possa ser posteriormente utilizado como um módulo em VHDL, é necessário definir todos os interfaces de entrada/saída. Por exemplo, para construir um módulo com o aspecto da fig. 3, podem-se definir as interfaces em Handel-C da seguinte maneira:

```
interface port_in(unsigned 4 DataL) ToL()
       with{std_logic_vector=1};
interface port_in(unsigned 4 DataR) ToR()
       with{std_logic_vector=1};
interface port_out() SegLeft(unsigned 7
                                SegSignalL=SegL)
       with{std_logic_vector=1};
interface port_out() SegRight(unsigned 7
                                SegSignalR=SegR)
       with{std_logic_vector=1};
interface port_out() EnL(unsigned 1
                                   EnLSignallL=1)
       with{std_logic_vector=0};
interface port_out() EnR(unsigned 1
                                   EnLSignalR=1)
       with{std_logic_vector=0};
```

O comando std_logic_vector=1 significa que o compilador vai criar um barramento do tamanho definido na variável de entrada/saída enquanto que std_logic_vector=0 o compilador cria apenas uma linha de sinal.

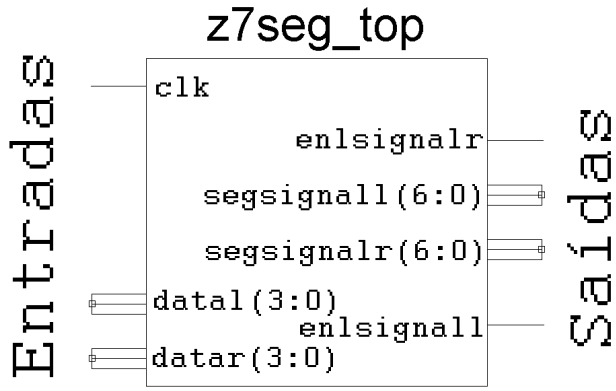

Fig. 3 – Módulo do descodificador de 7 segmentos para os dois displays

O ambiente *DK1 Design Suite* inclui também a possibilidade de compilar os programas escritos em Handel-C para a linguagem VHDL. Isto permite criar programas em Handel-C que depois de serem compilados para VHDL, podem ser utilizados no Xilinx ISE como módulos da biblioteca. Para tal, é necessário que no programa da Xilinx seja criada uma livraria Handel-C que inclua os ficheiros VHDL (Handel-C.vhd e xilroc.vhd) disponibilizados pela Celoxica como mostra a fig. 4 [5].

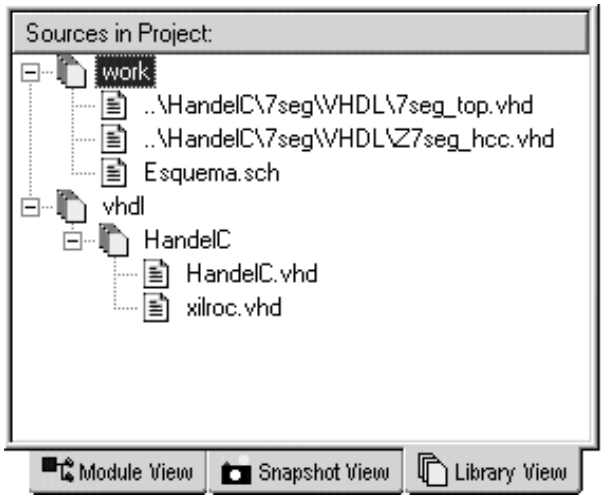

Fig. 4 – Livrarias do Xilinx ISE 5.1

VI. CONCLUSÕES

Deste artigo, pode-se verificar que se torna muito fácil a implementação de circuitos numa FPGA utilizando a uma linguagem semelhante ao ANSI-C (Handel-C) o que à partida seria extremamente complexo ser feito em VHDL.

É possível a utilização da linguagem Handel-C para a construção de módulos em VHDL o que permite a utilização destes módulos num projecto feito por exemplo no Xilinx ISE 5.1.

O projecto que inclui todas as partes consideradas acima está apresentado na WebCT [6,7].

AGRADECIMENTOS

Os autores agradecem ao Professor Valery Sklyarov pela ajuda prestada na elaboração deste artigo.

REFERÊNCIAS

[1] "RC100 Hardware Manual", Celoxica, 2001

[2] "Handel-C Language Reference Manual", Celoxica, 2002

[3] "RC100 Function Library Reference" Celoxica, 2001

[4] "RC100 Tutorials", Celoxica 2001

[5] "Xilinx 5 Software Manuals", Xilinx Inc, 2002

[6] Página *http://webct.ua.pt*, "1º Semestre" a disciplina "Sistemas Digitais Avançados".

[7] Página *http://webct.ua.pt*, "2º Semestre", a disciplina "Computação Reconfigurável".